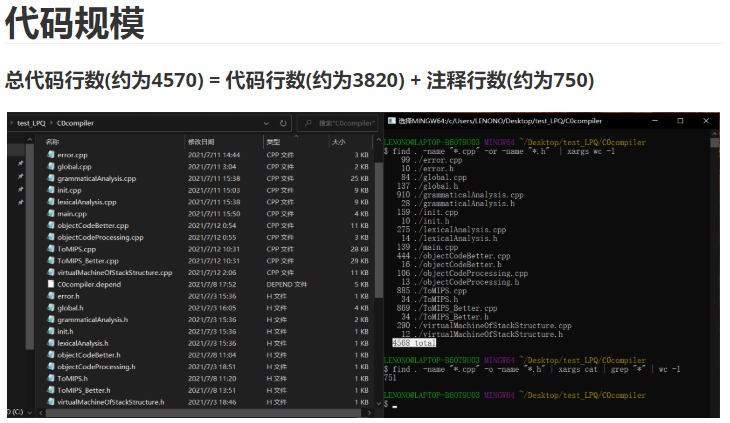
# 试点班报告

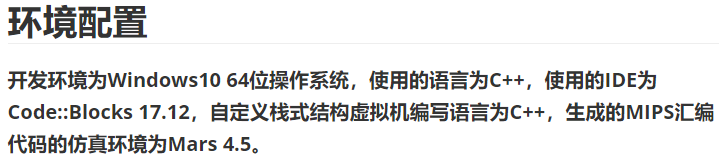
# 2018300102\_李沛琦

# 1、总体情况

# (1)代码规模:



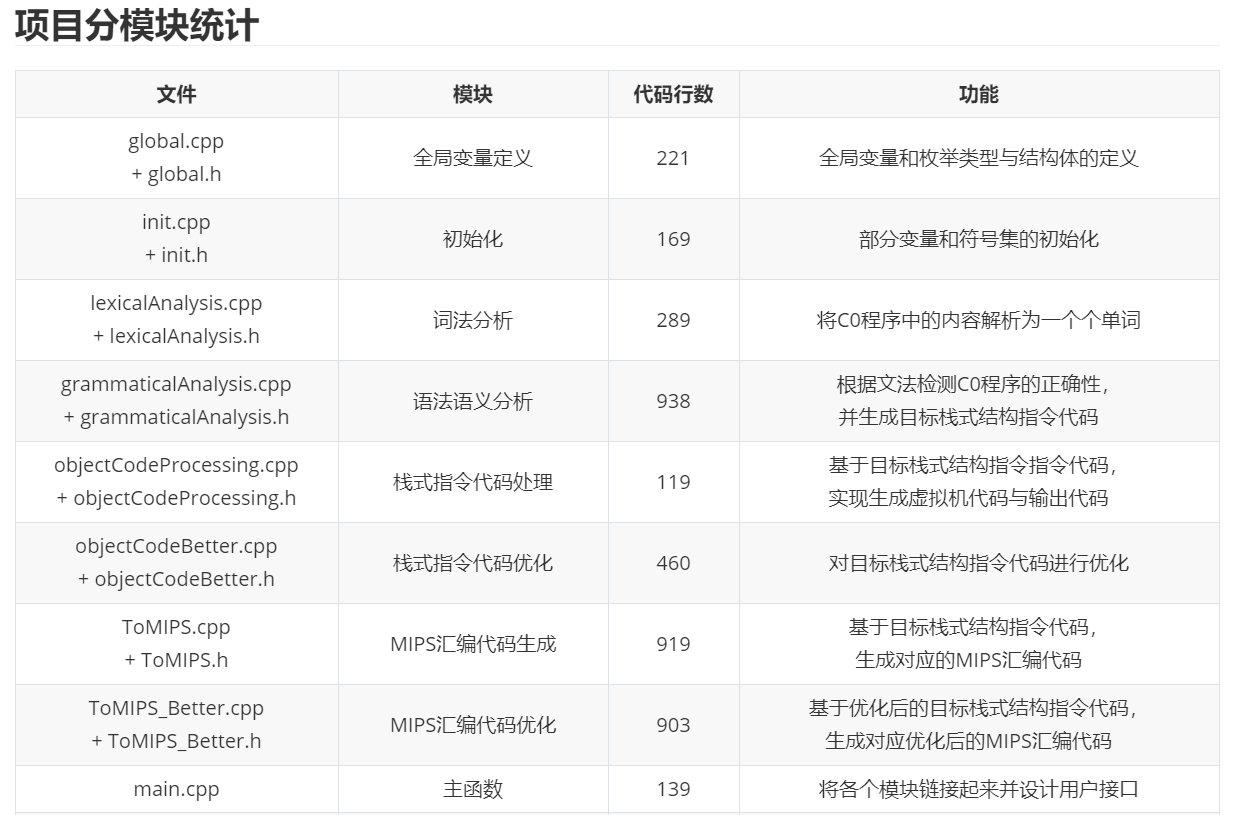
**(2)开发环境及说明：**



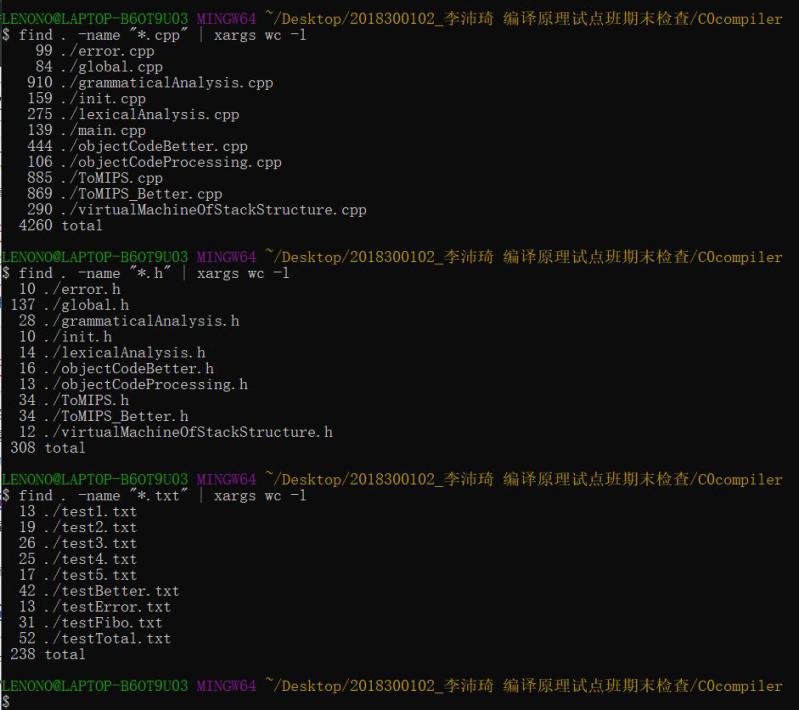
**(3)使用工具的情况**

未使用工具，采用递归下降的方式手写的前端。

**(4)项目分模块统计**







**(5)优化内容清单**

目前采用了效果明显且易于实现的窥孔优化，实现方式有：死代码删除、删除冗余的取和存、常量合并、代数化简几种优化方式优化方式。

由于采用的目标代码栈式结构指令代码在跨基本块操作过程的指令有明显特征，因此我通过控制窥孔窗口的大小来保证不会出现因为跨基本块进行优化而产生的错误。

另外，还可基于我实现的窥孔优化方式的框架，基于对应虚拟机偏好进一步采用常量传播、控制流优化、强度削弱、使用目标机惯用指令等优化方式，不过由于我才用的虚拟机没有设置明显运算或操作偏好，因此这后几种优化方式未在代码中体现，仅在注释中提供思路，详细的优化说明见后文优化部分。

# 2、语言定义

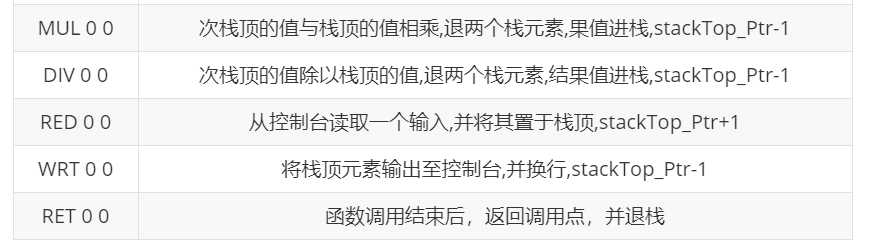
**(1)C0文法定义**



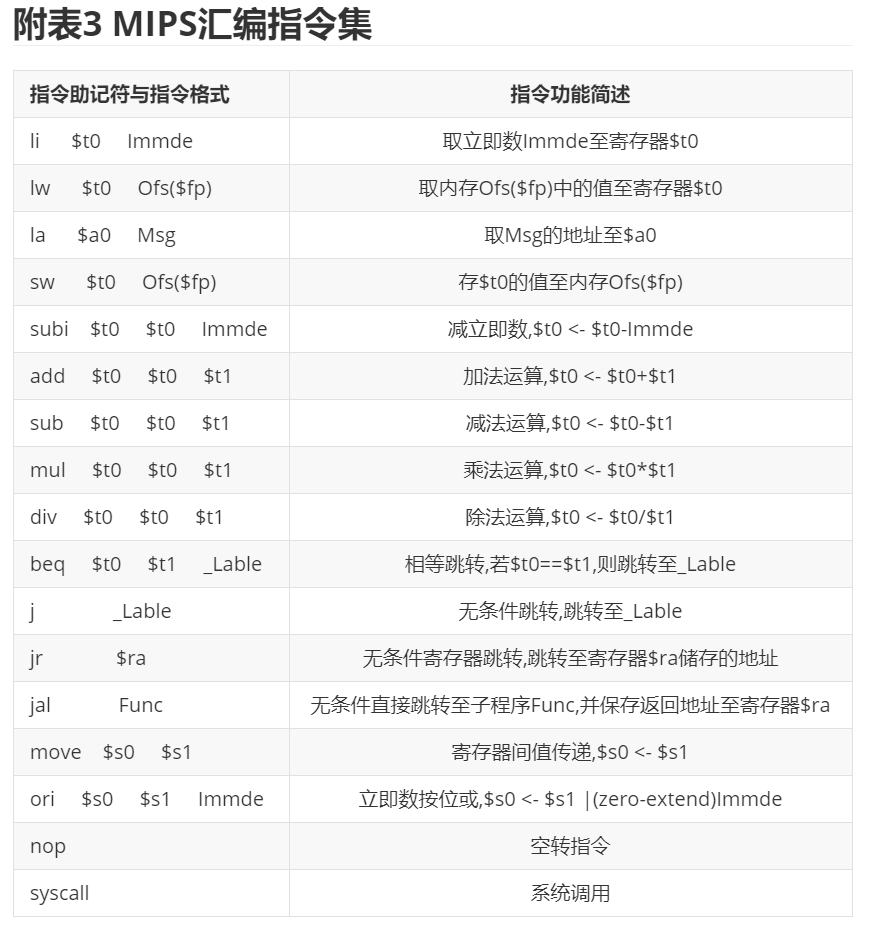


**(2)栈式指令定义**





**(3)MIPS汇编指令集**



# 3、语言文法的覆盖情况论述

**（1）部分文法举例说明**



分析：一个程序可以有变量说明和函数定义说明，也可以没有，如果都有的话就必须得遵从先声明变量再声明函数的顺序，且在程序中主函数是必须存在的。

**示例：**

1. Int a;
2. **void** test() {
3. }
4. **void** main() {
5. **int** a;
6. a=1;
7. printf(a);
8. }



分析：一个整数可以是一个无符号整数也可以是带正负号的无符号整数或者为0。

**示例：**

1. a = +10;
2. a = -10;
3. a = 0;
4. a = 10;



分析：标识符必须以字母开头或者下划线开头，后面可以跟数字或者字母或者下划线

**示例：**

1. **int** a;
2. **int** \_a;
3. Int abc\_123;



分析：复合语句类似于程序，也可以有变量说明，但是不可以有函数定义。

**示例：**

1. **void** test()
2. {
3. **int** a;
4. a = 1;
5. printf(a);
6. }



分析：main 函数必须用 void 修饰，不可以用 int 修饰。

**示例：**

1. **void** main()
2. {
3. **int** a;
4. a = 1;
5. printf(a);
6. }



分析：表达式前面可以带正负号来作用于整个表达式。

**示例：**

1. a = + (1+2\*3);
2. a = - (1+2\*3);



分析：if后括号内的表达式即为条件，表达式的值非0则表示条件为真，反之为假。

**示例：**

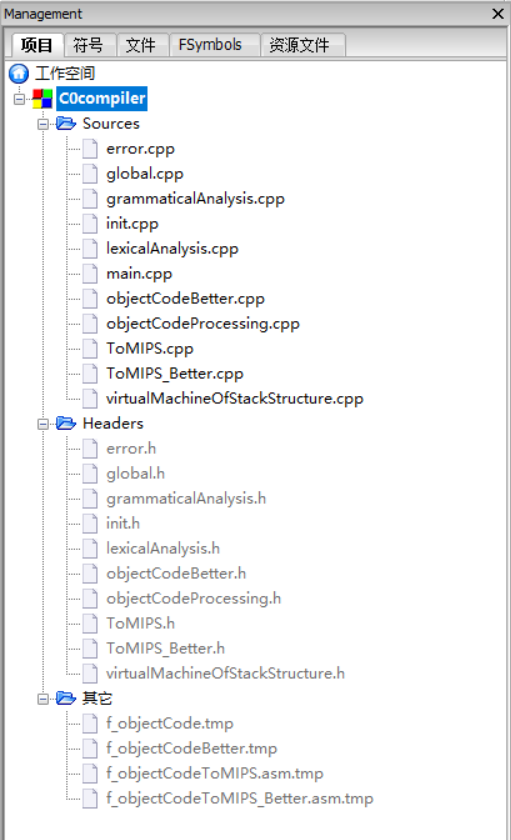
1. **if**(a) {
2. }
3. **if**(b+c) {
4. }
5. **else**{
6. }

## **（2）语法覆盖情况表**

语法覆盖内容的具体实现见测试用例与下文测试用例与测试结果中的阐述。

# 4、设计说明

## **（1）文件构成说明**



## （2）特性说明

编译器通过编译C0程序生成的栈式指令目标代码可以在源文件内自定义的栈式结构虚拟机上解释运行，生成的MIPS汇编代码可以在Mars上模拟仿真。C0compiler目录下为工程源代码和测试文件。

## （3）中间表示说明

采用无IR的方式，没有生成中间表示，直接生成了目标栈式结构指令代码，然后对生成的栈式结构指令代码进行优化，之后再通过栈式结构指令代码生成MIPS汇编代码。

## （4）栈式结构指令代码=>MIPS汇编代码的设计思路及说明

由于我自定义的栈式结构指令代码操作与内容已经十分接近MIPS汇编代码，因此大部分操作都可以对栈式结构指令代码进行线性扫描然后翻译并生成对应的MIPS汇编代码。

但是还是有一些操作不能直接通过单条栈式结构指令代码生成对应的MIPS汇编，于是我设置了一个窥孔来进行对栈式结构指令代码的线性扫描，根据上下文生成MIPS汇编代码，即可解决除while语句外的其他转化问题，详细内容见源码。

对while语句，分析得知，其栈式结构指令的特征为:while语句开始、条件判断、条件为假则跳转至while语句结束的有条件跳转、while函数体、无条件跳转至while语句开始。为此，在线性扫描栈式结构指令时，对每个有条件跳转指令的条件判断前都预写WhileStart的标记，在之后扫描到跳转至while语句开始的无条件跳转语句后，令其无条件跳转至对应的WhileStart标记。最后再对生成的MIPS汇编代码进行一边扫描，删除冗余的预写WhileStart标记，对正确匹配的while语句分配正确的跳转标记序号。

## （5）优化相关功能

目前采用了效果明显且易于实现的窥孔优化，实现方式有：死代码删除、删除冗余的取和存、常量合并、代数化简几种优化方式优化方式。

由于采用的目标代码栈式结构指令代码在跨基本块操作过程的指令有明显特征，因此我通过控制窥孔窗口的大小来保证不会出现因为跨基本块进行优化而产生的错误。

另外，还可基于我实现的窥孔优化方式的框架，基于对应虚拟机偏好进一步采用常量传播、控制流优化、强度削弱、使用目标机惯用指令等优化方式，不过由于我才用的虚拟机没有设置明显运算或操作偏好，因此这后几种优化方式未在代码中体现，仅在注释中提供思路。

基于目标栈式结构指令代码的优化过程需要计算优化后需要修正的偏移量，从而保证跳转语句的正确性，具体的优化内容如下，更多详细内容见源代码。

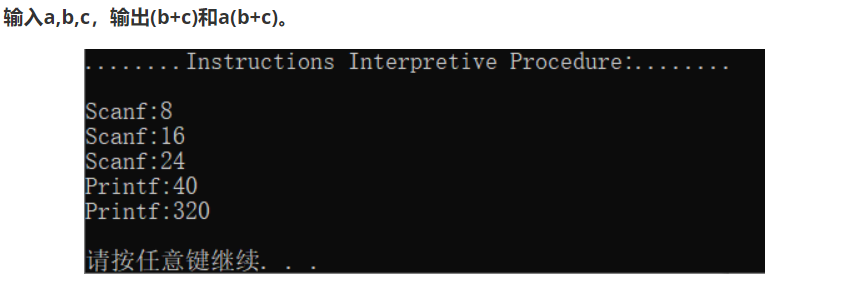


## （10）测试用例与测试结果

编写了几个自定义测试用例：test1.txt、test2.txt、test3.txt、test4.txt、test5.txt、testFibo.txt、testTotal.txt、testError.txt、testBetter.txt，以下大致说明一下每个测试用例的内容与用途和一组测试结果正确的输入输出。

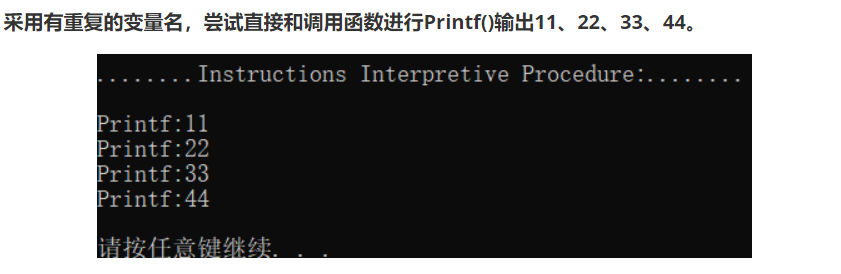
**test1.txt:**

1. **void** main ()
2. {
3. **int** a,b,c;
4. scanf(a);
5. scanf(b);
6. scanf(c);
7. c=b+c;
8. printf(c);
9. a=a\*c;
10. printf(a);
11. }



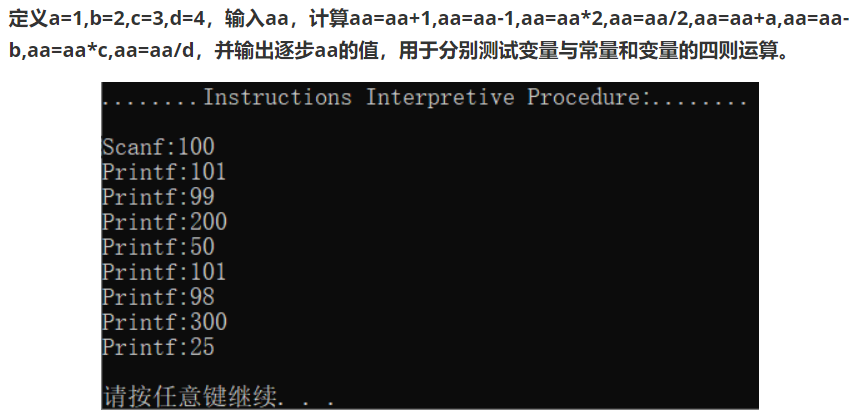
**test2.txt:**

1. **void** fun(){
2. **int** a;
3. a=44;
4. printf(a);
5. }
7. **void** main(){
8. **int** a,b,c;
9. a=11;
10. printf(a);
11. b=22;
12. printf(b);
13. c=a+b;
14. printf(c);
15. fun();
16. }



**test3.txt:**

1. **void** main(){
2. **int** a,b,c,d,e,aa;
3. scanf(a);
4. b=1;
5. c=2;
6. d=3;
7. e=4;
8. aa=a+1;
9. printf(aa);
10. aa=a-1;
11. printf(aa);
12. aa=a\*2;
13. printf(aa);
14. aa=a/2;
15. printf(aa);
16. aa=a+b;
17. printf(aa);
18. aa=a-c;
19. printf(aa);
20. aa=a\*d;
21. printf(aa);
22. aa=a/e;
23. printf(aa);
24. }



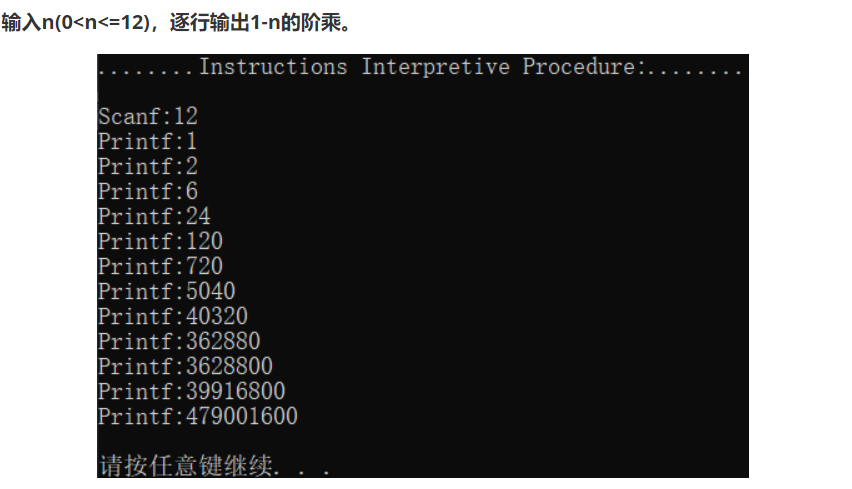
**test4.txt:**

1. **void** f1(){
2. **int** a;
3. a=1;
4. printf(a);
5. }
6. **void** f2(){
7. **int** a;
8. a=2;
9. printf(a);
10. }
11. **void** f3(){
12. **int** a;
13. a=3;
14. printf(a);
15. }
16. **void** main(){
17. **int** a;
18. a=4;
19. f1();
20. f2();
21. f3();
22. printf(a);
23. }



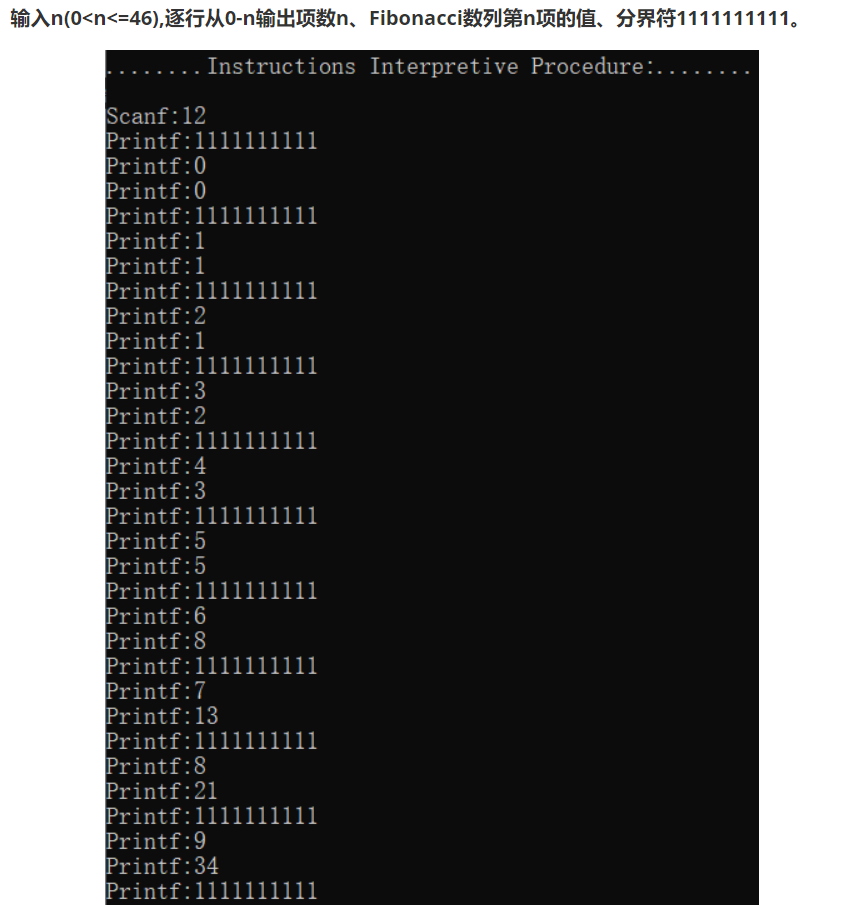
**test5.txt:**

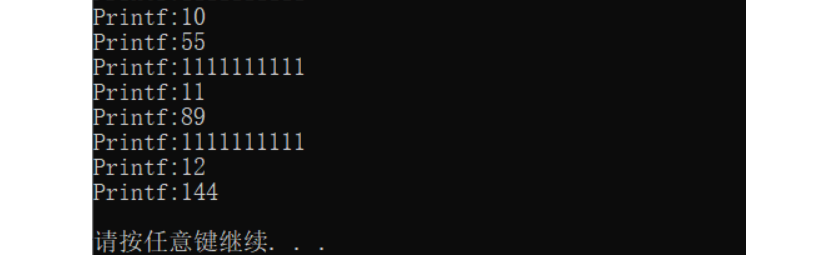
1. **void** main (){
2. **int** a,i,sum;
3. scanf(a);
4. i=a;
5. a=1;
6. sum=1;
8. **while**(i){
9. sum = sum\*a;
10. printf(sum);
11. a=a+1;
12. i=i-1;
13. }
14. }



**testFibo.txt:**

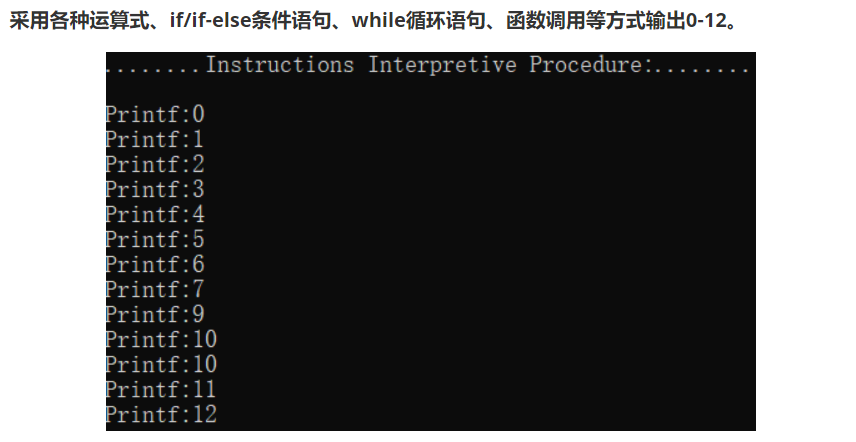
1. **void** main ()
2. {
3. **int** i,a,b,tmp,N,PL;
4. i = 0;
5. a = 0;
6. b = 1;
7. PL = 1111111111;
8. scanf(N);
9. printf(PL);
10. printf(i);
11. printf(a);
12. i = i+1;
13. printf(PL);
14. printf(i);
15. printf(b);
16. i = i+1;
17. N = N-1;
18. **while**(N)
19. {
20. tmp = a+b;
21. a = b;
22. b = tmp;
23. printf(PL);
24. printf(i);
25. printf(b);
26. i = i+1;
27. N = N-1;
28. }
29. }





**testTotal.txt:**

1. **void** f8(){
2. **int** a;
3. a=9;
4. printf(a);
5. }
6. **void** f9(){
7. **int** a;
8. a=10
9. printf(a);
10. }
12. **void** main(){
13. **int** a,b,c,i,A,B,C,Aa;
14. a=1;
15. b=2;
16. c=4;
17. i=2;
18. **if**(a){
19. printf(0);
20. printf(1);
21. }
22. **else**{
23. printf(99);
24. }
25. a = a - 1;
26. **if**(a){
27. printf(99);
28. }
29. **while**(b){
30. printf(i);
31. i=i+1;
32. b=b-1;
33. }
34. **while**(c){
35. printf(i);
36. i=i+1;
37. c=c-1;
38. }
39. f8();
40. f9();
41. A=11;
42. B=12;
43. C=1;
44. Aa=A-1;
45. printf(Aa);
46. Aa=A\*C;
47. printf(Aa);
48. Aa=B/C;
49. printf(Aa);
50. }



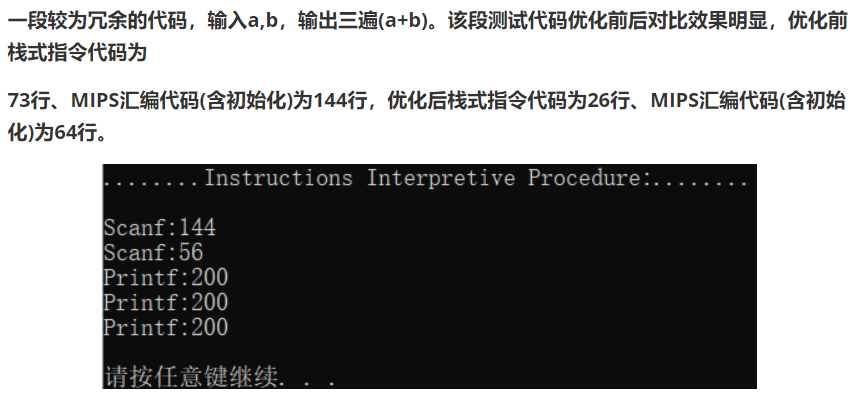
**testError.txt:**

1. **void** main()
2. {
3. **int** m,a;
4. scanf(m;
5. a=3;
6. **while**(a)
7. {
8. printf(m)
9. a=a-1;
10. }
11. }



**testBetter.txt:**

1. **void** main ()
2. {
3. **int** a,b,c;
4. c=0+0;
5. c=c+0;
6. c=c-0;
7. c=0+c;
8. c=c\*1;
9. c=c/1;
10. c=1\*c;
12. scanf (a);
13. c=a;
14. a=c;
15. scanf (b);
16. c=b;
17. b=c;
19. **while**(0)
20. {
21. printf(0);
22. printf(9);
23. }
25. **if**(0)
26. {
27. printf(0);
28. printf(9);
29. }
31. c=a+b;
32. a=c;
33. c=a;
34. b=c;
35. c=b;
37. printf (a);
38. printf (b);
39. printf (c);
40. }



**优化前的目标栈式结构指令代码(74行)：**

1. (00) JMP 0 1
2. (01) INT 0 6
3. (02) LIT 0 0
4. (03) LIT 0 0
5. (04) ADD 0 0
6. (05) STO 0 5
7. (06) LOD 0 5
8. (07) LIT 0 0
9. (08) ADD 0 0
10. (09) STO 0 5
11. (10) LOD 0 5
12. (11) LIT 0 0
13. (12) SUB 0 0
14. (13) STO 0 5
15. (14) LIT 0 0
16. (15) LOD 0 5
17. (16) ADD 0 0
18. (17) STO 0 5
19. (18) LOD 0 5
20. (19) LIT 0 1
21. (20) MUL 0 0
22. (21) STO 0 5
23. (22) LOD 0 5
24. (23) LIT 0 1
25. (24) DIV 0 0
26. (25) STO 0 5
27. (26) LIT 0 1
28. (27) LOD 0 5
29. (28) MUL 0 0
30. (29) STO 0 5
31. (30) RED 0 0
32. (31) STO 0 3
33. (32) LOD 0 3
34. (33) STO 0 5
35. (34) LOD 0 5
36. (35) STO 0 3
37. (36) RED 0 0
38. (37) STO 0 4
39. (38) LOD 0 4
40. (39) STO 0 5
41. (40) LOD 0 5
42. (41) STO 0 4
43. (42) LIT 0 0
44. (43) JPC 0 49
45. (44) LIT 0 0
46. (45) WRT 0 0
47. (46) LIT 0 9
48. (47) WRT 0 0
49. (48) JMP 0 42
50. (49) LIT 0 0
51. (50) JPC 0 55
52. (51) LIT 0 0
53. (52) WRT 0 0
54. (53) LIT 0 9
55. (54) WRT 0 0
56. (55) LOD 0 3
57. (56) LOD 0 4
58. (57) ADD 0 0
59. (58) STO 0 5
60. (59) LOD 0 5
61. (60) STO 0 3
62. (61) LOD 0 3
63. (62) STO 0 5
64. (63) LOD 0 5
65. (64) STO 0 4
66. (65) LOD 0 4
67. (66) STO 0 5
68. (67) LOD 0 3
69. (68) WRT 0 0
70. (69) LOD 0 4
71. (70) WRT 0 0
72. (71) LOD 0 5
73. (72) WRT 0 0
74. (73) RET 0 0

**优化后的目标栈式结构指令代码(27行)：**

1. (00) JMP 0 1
2. (01) INT 0 6
3. (02) LIT 0 0
4. (03) STO 0 5
5. (04) RED 0 0
6. (05) STO 0 3
7. (06) LOD 0 3
8. (07) STO 0 5
9. (08) RED 0 0
10. (09) STO 0 4
11. (10) LOD 0 4
12. (11) STO 0 5
13. (12) LOD 0 3
14. (13) LOD 0 4
15. (14) ADD 0 0
16. (15) STO 0 5
17. (16) LOD 0 5
18. (17) STO 0 3
19. (18) LOD 0 5
20. (19) STO 0 4
21. (20) LOD 0 3
22. (21) WRT 0 0
23. (22) LOD 0 4
24. (23) WRT 0 0
25. (24) LOD 0 5
26. (25) WRT 0 0
27. (26) RET 0 0

**优化前的MIPS汇编代码(144行)：**

1. .data
2. Enter: .asciiz  "\n"
3. .text
4. ori     $fp     $sp     0
5. li      $t9     0x7fffeffc  #global stack bottom
6. li      $t8     0x10010000  #save **word**
7. j   \_\_main
9. \_\_main:
10. #Save Register
11. sw      $s0     0($sp)
12. subi    $sp     $sp     4
13. sw      $fp     0($sp)
14. subi    $sp     $sp     4
15. add     $fp     $sp     $0
16. sw      $ra     ($sp)
17. subi    $sp     $sp     4
18. #Save Register Done!
20. li      $t0     0
21. sw      $t0     ($sp)
22. subi    $sp     $sp     4
23. li      $t0     0
24. sw      $t0     ($sp)
25. subi    $sp     $sp     4
26. li      $t0     0
27. sw      $t0     ($sp)
28. subi    $sp     $sp     4
30. li      $t0     0
31. li      $t1     0
32. add     $t0     $t0     $t1
33. sw      $t0     -12($fp)
34. lw      $t0     -12($fp)
35. li      $t1     0
36. add     $t0     $t0     $t1
37. sw      $t0     -12($fp)
38. lw      $t0     -12($fp)
39. li      $t1     0
40. sub     $t0     $t0     $t1
41. sw      $t0     -12($fp)
42. li      $t0     0
43. lw      $t1     -12($fp)
44. add     $t0     $t0     $t1
45. sw      $t0     -12($fp)
46. lw      $t0     -12($fp)
47. li      $t1     1
48. mul     $t0     $t0     $t1
49. sw      $t0     -12($fp)
50. lw      $t0     -12($fp)
51. li      $t1     1
52. **div**     $t0     $t0     $t1
53. sw      $t0     -12($fp)
54. li      $t0     1
55. lw      $t1     -12($fp)
56. mul     $t0     $t0     $t1
57. sw      $t0     -12($fp)
58. li      $v0     5
59. syscall
60. sw      $v0     -4($fp)
61. lw      $t0     -4($fp)
62. sw      $t0     -12($fp)
63. lw      $t0     -12($fp)
64. sw      $t0     -4($fp)
65. li      $v0     5
66. syscall
67. sw      $v0     -8($fp)
68. lw      $t0     -8($fp)
69. sw      $t0     -12($fp)
70. lw      $t0     -12($fp)
71. sw      $t0     -8($fp)
72. \_\_WhileStart0:
73. li      $t0     0
74. li      $t1     0
75. beq     $t0     $t1     \_\_Lable0
76. li      $a0     0
77. li      $v0     1
78. syscall
79. li      $v0     4
80. la      $a0     Enter
81. syscall
82. li      $a0     9
83. li      $v0     1
84. syscall
85. li      $v0     4
86. la      $a0     Enter
87. syscall
88. j               \_\_WhileStart0
90. \_\_Lable0:
91. li      $t0     0
92. li      $t1     0
93. beq     $t0     $t1     \_\_Lable1
94. li      $a0     0
95. li      $v0     1
96. syscall
97. li      $v0     4
98. la      $a0     Enter
99. syscall
100. li      $a0     9
101. li      $v0     1
102. syscall
103. li      $v0     4
104. la      $a0     Enter
105. syscall
107. \_\_Lable1:
108. lw      $t0     -4($fp)
109. lw      $t1     -8($fp)
110. add     $t0     $t0     $t1
111. sw      $t0     -12($fp)
112. lw      $t0     -12($fp)
113. sw      $t0     -4($fp)
114. lw      $t0     -4($fp)
115. sw      $t0     -12($fp)
116. lw      $t0     -12($fp)
117. sw      $t0     -8($fp)
118. lw      $t0     -8($fp)
119. sw      $t0     -12($fp)
120. lw      $a0     -4($fp)
121. li      $v0     1
122. syscall
123. li      $v0     4
124. la      $a0     Enter
125. syscall
126. lw      $a0     -8($fp)
127. li      $v0     1
128. syscall
129. li      $v0     4
130. la      $a0     Enter
131. syscall
132. lw      $a0     -12($fp)
133. li      $v0     1
134. syscall
135. li      $v0     4
136. la      $a0     Enter
137. syscall
139. \_\_FEND\_LAB\_1:
140. lw      $ra     0($fp)
141. lw      $sp     8($fp)
142. lw      $fp     4($fp)
143. li      $v0     10
144. syscall

**优化后的MIPS汇编代码(64行)：**

1. .data
2. Enter: .asciiz  "\n"
3. .text
4. ori     $fp     $sp     0
5. li      $t9     0x7fffeffc  #global stack bottom
6. li      $t8     0x10010000  #save **word**
7. j   \_\_main
8. \_\_main:
9. #Save Register
10. sw      $s0     0($sp)
11. subi    $sp     $sp     4
12. sw      $fp     0($sp)
13. subi    $sp     $sp     4
14. add     $fp     $sp     $0
15. sw      $ra     ($sp)
16. subi    $sp     $sp     4
17. #Save Register Done!
18. li      $t0     0
19. sw      $t0     ($sp)
20. subi    $sp     $sp     12
21. li      $t0     0
22. sw      $t0     -12($fp)
23. li      $v0     5
24. syscall
25. sw      $v0     -4($fp)
26. lw      $t0     -4($fp)
27. sw      $t0     -12($fp)
28. li      $v0     5
29. syscall
30. sw      $v0     -8($fp)
31. lw      $t0     -8($fp)
32. sw      $t0     -12($fp)
33. lw      $t0     -4($fp)
34. lw      $t1     -8($fp)
35. add     $t0     $t0     $t1
36. sw      $t0     -12($fp)
37. lw      $t0     -12($fp)
38. sw      $t0     -4($fp)
39. lw      $t0     -12($fp)
40. sw      $t0     -8($fp)
41. lw      $a0     -4($fp)
42. li      $v0     1
43. syscall
44. li      $v0     4
45. la      $a0     Enter
46. syscall
47. lw      $a0     -8($fp)
48. li      $v0     1
49. syscall
50. li      $v0     4
51. la      $a0     Enter
52. syscall
53. lw      $a0     -12($fp)
54. li      $v0     1
55. syscall
56. li      $v0     4
57. la      $a0     Enter
58. syscall
59. \_\_FEND\_LAB\_1:
60. lw      $ra     0($fp)
61. lw      $sp     8($fp)
62. lw      $fp     4($fp)
63. li      $v0     10
64. syscall